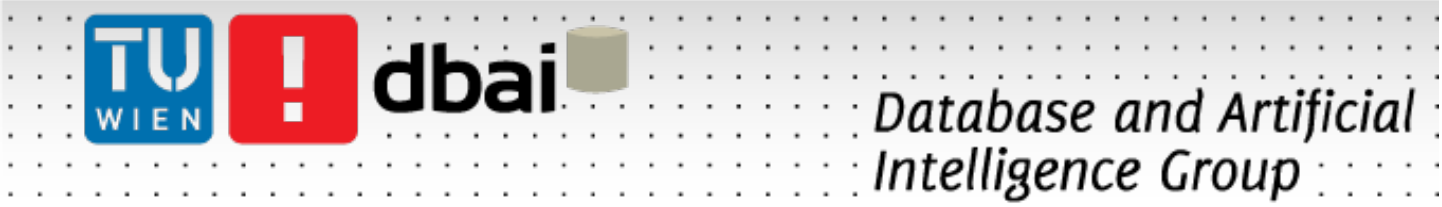

Exercise 2

(15 points)

The goal in this year's exercise is to develop an XML-based format to represent data for an auction house (like eBay). In Exercise 1 you will create an XML Schema and XML document and in Exercise 2 you will be querying and transforming this document with XML-related technologies.

An auction house offers products. A user can place a bid on a specific product. After a set date, the auction for a product ends. The user with the highest bid wins the product.

Remark: The following data format can be seen as a database for an auction house.

This exercise consists of three parts. First, we will use DOM and SAX to access and modify XML documents. Second, we will evaluate an xQuery expression over an XML document. And third, we will use XSLT to transform the XML document into a Webpage.

Template

In the following ZIP-file you will find a general template for this exercise. This template also contains an ant-script (build.xml) which can be used to test your solution. A similar ant-script (with more test cases) will be used during the assignment discussions.

- Template SSD exercise 2 - release 1

Please, remember:

- It is **mandatory** that you use this template!
- All files and paths in this description refer to this template.
- Precise instructions for each ant target can be found in the sections below.
- We will update the template with an XML document and a schema file (the solution of Exercise 1) after the assignment discussions of Exercise 1. These files can then be used as input for the following assignments. You can also use your solution.

DOM and SAX

Imagine that your application is divided into a client that displays the auction house and a server that processes and manages the auction house. The auction house displays its product on a webpage (the client). The user chooses a product and places a bid. The client sends the bid to the server, which stores and processes the auction house data (in an XML document). The client uses the following "bid" XML document for this communication:

```
<bid product="2">
   <user>Lisa</user>
   <bid>500</bid>
</bid>
```

This XML document states that the user "Lisa" placed a bid of "500" on product with id "2".

Your goal is to manipulate the `auctions.xml` document with DOM in such a way, that the above "bid" is stored in the document. The "bid" document is parsed using SAX.

You have to ensure that your resulting document validates against the given `auctions.xsd`. Additionally, you have to check if the placed bid fulfills the following criteria:

- Bids can only be placed for products in the document.

- The auction has neither *ended* (`auctionDate` is before the currentDate) nor is *expired* (auction doesn't have an `expired` child element).
- The placed bid must be larger than the bids already placed for the product.
- The placed bid must be smaller than the current balance of the user.

Description of Classes

The template provides you two classes. The class `SSD` contains the main program logic. The class `Bidding Handler` is used as a SAX handler for parsing the "bid" document and manipulating the auction house document. A detailed description follows:

- *Class:* **SSD**
  - *Variables:*
    - `static DocumentBuilderFactory documentBuilderFactory`: stores an instance of a document builder factory.
    - `static DocumentBuilder documentBuilder`: stores an instance of a document builder.
  - *Methods:*
    - `static void main(String [] args) throws Exception`: Entry point for the program. Parses the program arguments and calls `initialize` and `transform`.
    - `static void initialize() throws Exception`: Initializes the `documentBuilderFactory` and the `documentBuilder` variables.
    - `static void transform(String inputPath, String bidPath, String outputPath) throws Exception`: **Your task is to implement this method**. First, you have to create a Document from the filename given in the `inputPath` variable. Second, you have to set up the SAX parser, in order to parse the XML document given in the `bidPath` variable. For this you have to create an instance of the `BiddingHandler` class, which needs the previously created Document as an argument to its constructor. Third, you parse the bid document. The `BiddingHandler` will change the document. Last, retrieve this document with the method `getDocument()` and store this document in the file given by the `outputPath` variable.
    - `static void exit(String message)`: You can use this method to output an error and exit the program.
- *Class:* **BiddingHandler**
  - *Variables:*
    - `static XPath xPath`: use this XPath instance to compile xPath queries that can be evaluated over the document or specific context nodes.
    - `Document auctionsDoc`: stores the DOM document of the auction house XML file.
    - `String eleText`: stores the text content of XML elements.
    - **You can specifiy additional variables in this class.**
  - *Methods:*
    - `BiddingHandler(Document doc)`: The constructor which has a Document variable as argument.
    - `void characters(char[] text, int start, int length)`: SAX calls this method, when it sees character data. This character data is stored in the `eleText` variable.
    - `Document getDocument()`: Returns the XML document stored in this class.
    - **Specify additional methods to parse the "bid" document (e.g.: `startElement`, etc.) and modify the `auctionsDoc` document.**

Call of program and result

The code implemented in `src/ssd/SSD.java` can be run using three program arguments. The first is an auction house document as input (e.g. `auctions.xml` from exercise 1 or the file located in the `resources` folder). The second is a "bid" document (e.g. `resources/auction-bid.xml`). The last

is an auction house document as output (e.g. `output/auctions.xml`). We have configured two ant-targets:

- `ant run-dry`:
  applies the "bid" document `resources/auctions-bid.xml` to the auction house document `resources/auctions.xml` and stores the output to `output/auctions.xml`.
- `ant run-persistent`:
  applies the "bid" document `resources/auctions-bid.xml` to the auction house document `resources/auctions.xml` and stores the output to `resources/auctions.xml`.
  Therefore, the input file will be overwritten and the "bid" is made persistent.

After the assignment discussion, we will add a file `resources/auctions-sample-out.xml` that shows the output of the program by applying `resources/auction-bid.xml` to `resources/auctions.xml`.

Hints
You can evaluate XPath expressions over an XML document (or also relative to a node) using the following statements:

```
XPathExpression xpathExpr = xPath.compile("//users");
NodeList playerList = (NodeList)xpathExpr.evaluate(auctionsDoc,
                                                   XPathConstants.NODESET);
```

Summary

- **Resulting Files**: `SSD.java` and `BiddingHandler.java`
- **Total points**: 10

XQuery

In this exercise we will create an XQuery that evaluated over an auction house document, will give you a list of users together with the won bids.
The output should look as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<won>
   <user name="Bart">
      <product value="450">Web Engineering</product>
      <product value="150">XML in a Nutshell</product>
   </user>
   <user name="Homer"/>
   <user name="Lisa">
      <product value="300">The Great Gatsby (Movie + Soundtrack)</product>
   </user>
</won>
```

The meaning of the document is as follows: The user "Bart" won the products "Web Engineering" (with value "450") and "XML in a Nutshell" (with value "150). The user "Homer" has not won any products. The user "Lisa" won the products "The Great Gatsby (Movie + Soundtrack)" (with value "300").

The elements won should be ordered by user ascending and the elements product by value descending.

You can write this query in `src/xquery.xq`. The query can be evaluated over `resources/auctions.xml` using the following command:

- `ant run-xquery`

The output is written to `output/xquery-out.xml`.

Summary

- **Resulting File**: `xquery.xq`
- **Points**: 2

XSLT

Create an XSLT stylesheet `src/auctions.xsl`, which transforms a (for the schema `resources/auctions.xsd`) valid XML document (like `resources/auctions.xml`) into an HTML document specified as follows: The HTML document should give an overview of the products contained in an auction house `auctions.xml` document. For each product give the name and the description. Also list the total number of bids and each bid for the product. The bid with the highest value should be in red color. An example output is given here.

Call of program and result

The transformation using the stylesheet in `src/auctions.xsl` from `resources/auctions.xml` to `output/auctions.html` can be run using the following command:

- `ant run-xslt`

The output is written to `output/auctions.html`.

Summary

- **Resulting File**: `auctions.xsl`
- **Points**: 3

Upload

In order to create a zip file `ssd-exercise2-ss16.zip` you have to use the following command:

- `ant zip`

This zip file has to be uploaded until 08.06.2016 23:55 in TUWEL . We will grade the last uploaded solution.

Assignment discussion

You can receive at most 15 points for Exercise 3. During the assignment discussion we will not only check your solution for correctness, but will also ask some questions regarding the used technologies.

To obtain the full number of points your solution has to be solved correctly and you have to be able to explain it. Copied solutions are awarded 0 points!

In your own interest come **ontime** to your assignment discussion or else we don't guarantee that your solution is completely checked in the remaining time of the reserved slot.